*Neurogammon*

Harrison Bonner   
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
hbonner@iu.edu  
  
Truman Brubaker  
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
trubruba@iu.edu  
  
Zachary Caswell  
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
zcaswell@iu.edu  
  
Ahmet Cengiz   
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
ahcengiz@iu.edu

Sabrina Chowdhury   
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
sabchow@iu.edu

Foster Clinton  
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
fclinton@iu.edu

Matthew Cobbs  
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
cobbsma@iu.edu

Nick Derf  
Computer Science Dept.  
Indiana University Purdue University IndianapolisIndianapolis, United States of America  
nderf@iu.edu

*Abstract*—Neurogammon 1.0 was a groundbreaking achievement when it won the First Computer Olympiad in London, England, nineteen eighty-nine. The purpose of this paper is to provide a comprehensive review of the research conducted by our group into how a multi-layer neural network that uses back propagation, similar to Neurogammon, operates. Concluding remarks and possible directions of growth are also discussed at the end.

Keywords—Neural Network, Game Theory, Artificial Intelligence, Backgammon, Computer games, Learning Systems

# Introduction

Backgammon at the basic level is a game where two adversarial players take turn rolling dice and moving their pieces. The game is won once a player has moved all their pieces off the board. The game however becomes complex after considering how the game allows one player to send opponent pieces back to the other end of the board. That is the defining rule to backgammon that causes human players to use advanced strategies while playing [1]. Due to the highly nondeterministic nature of the game, it is a good candidate for a neural network as an artificial intelligence opponent.

# History And Background

Gerald Tesauro is currently a Principal Research Staff Member in AI Science at IBM’s TJ Watson Research Center. He is also a Fellow of the AAAI, a Fellow of the ACM, a member of the Board of Directors of the Neural Information Processing Systems Foundation, and an Associate Editor of the ICGA Journal. He is most famous for his AI Neural Network that learned how to play Backgammon.

He first published in 1986 an article called “Simple Neural Models of Classical Conditioning” which then paved the way for his 1987 publication on Backgammon. Continuing onto 1989, this is the year that Tesauro’s AI won the 1st Computer Olympiad in London after defeating all opponents in Backgammon. At the time, the computer was able to play at an intermediate level, in terms of human capabilities.

The first version of the AI, which was called Neurogammon, consisted of a neural network who trained by using backpropagation on a pre-recorded data set of expert games. Its inputs included the board information and the set of features. This was the version that won in the competition in London.

Tesauro then went on to create another version of this AI, called TD-Gammon. This instead used reinforcement learning through multilayer neural networks. They were trained by Temporal Difference (TD) to learn more complex nonlinear functions.

His main field of interest is neural networks and modeling data, even still to this day. One of his most recent publications in 2018 is, again, about Neural Networks and it is titled: “R^3: Reinforced Ranker-Reader for Open-Domain Question Answering.”

# Neural Network Training

## Introduction To Neural Networks

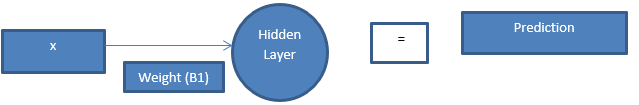
Neural networks endeavor to recognize relationships between a set of data through a process. It is designed in such a way that it can mimic biological neurons. In other words, it has simulated neurons that replicate human thinking processes. Neural networks are generally multilayer to classify series of data. It generally has 3 layers: Input Layer, Hidden Layer, Output Layer. Consider the single neuron pictured in figure 1, X is our input layer and B1 is the slop estimator of logistic regression. In the hidden layer, we use the sigmoid function for activation. Assuming B0 is the network’s bias, the outputted predicted probability is equal to the sigmoid function of B1\*x+B0

Figure 1- Single Neural Network Neuron

When this neuron is expanded for multilayer, the network looks something like Figure 2, there are 2 inputs and 2 neurons to support the inputs, and WX,Y is a weight associated with input X and neuron Y. The output of the hidden layer will be calculated as: Z1=W1,1\*In1+W2,1\*In2+bias\_n1. Neuron 1 activation =sigmoid (Z1). So, we can summarize this as:

[W1,1 W1,2 \* [x1+ [b1 = [Z1 W2,1 W2,2] x2] b2] z2]

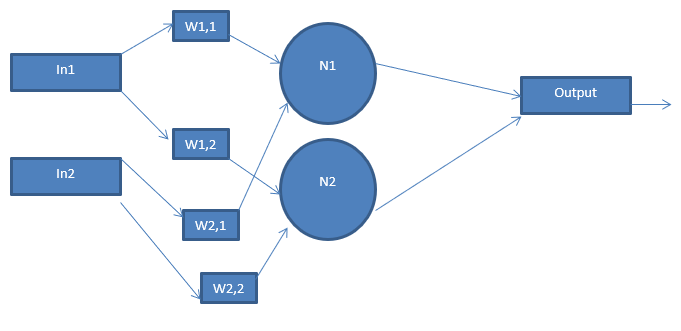


Figure 2- Multilayer Neural Network

## Training Data

For our neural network to function, we must first train it from some generated data. Our model utilizes 30 features: number of checkers in each position of the board (24 features), dice roll (2 features, 1 per die), number of checkers in jail (2 features, 1 per player), number of checkers in the home (2 features, 1 per player). Our data originated by capturing the 30 features for some observed games. We accounted for the adversarial nature of the game by using negative numbers for the opponent player.

## Neural Network Layout

As described in the multilayer neural network introduction, our neural network will consist of 31 inputs (one for each feature) and 5 hidden layers. The output of the network will classify using the sigmoid function, outputting a binary 0 for current player loss, or 1 for current player win.

## Self-Learning

For this project, the game backgammon was written in python and a simple GUI developed (pictured in Figure 3) in the console to observe that the game behaved as expected.   Control of each player was given to a separate agent.

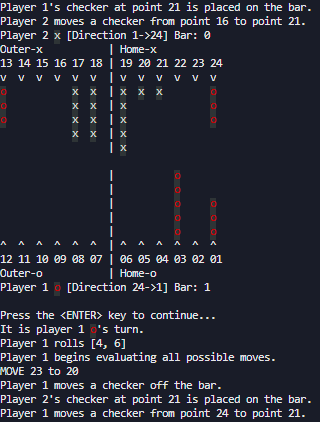


Figure 3- Self-Testing GUI

Data was generated by having the two agents play each other continuously.  Each time a dice was rolled, the agent would evaluate each resultant board state from all possible moves.  Each move was given a different weight determined by the aggregate of how many times the resultant board state led to victory or defeat.  The move with the highest chance of victory was chosen as the next move.

At the conclusion of a game, training data was either created (if none existed for that board state) or updated with the outcome of that game.  During the training process millions of different board states were recorded.  The data for each individual board state was placed into its own bucket (file), such that necessary records could be instantly retrieved and used for comparison as needed.

With each successive playthrough each agent gained more data into which moves were the most optimal.  The goal is to have a fully trained agent, with millions of games played, square off against an agent with no training data.  The expected outcome is that the trained agent would win most of these matches.

##### References

1. Tesauro, Gerald. “Practical Issues in Temporal Difference Learning.” Reinforcement Learning, vol. 8, 1992, pp. 33–53., doi:10.1007/BF00992697

] G. Tesauro, "Neurogammon: a neural-network backgammon program," 1990 IJCNN International Joint Conference on Neural Networks, San Diego, CA, USA, 1990, pp. 33-39 vol.3.